

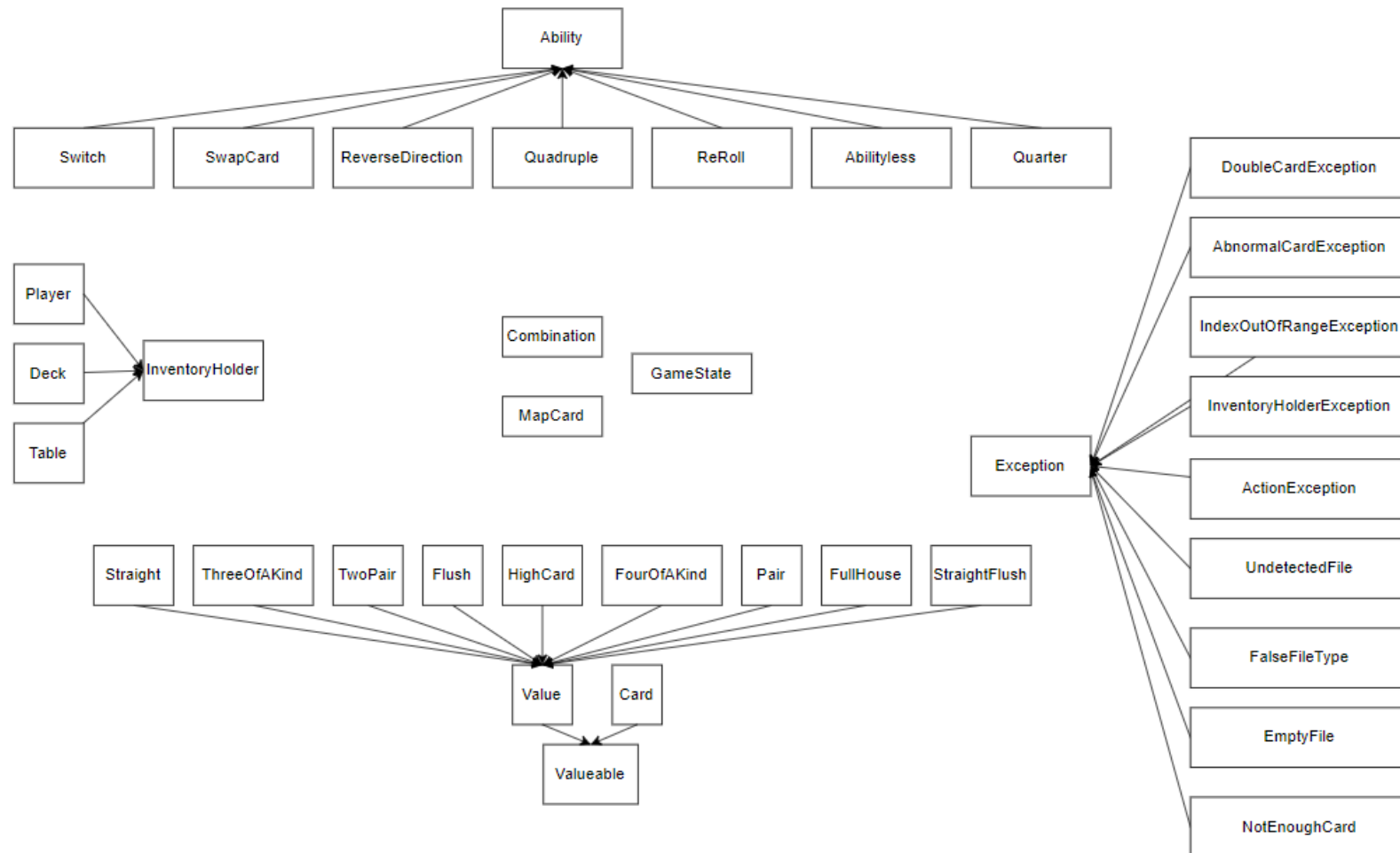
Kode Kelompok : 911
Nama Kelompok : ninuninu
1. 13521009 / Christophorus Dharma Winata
2. 13521018 / Syarifa Dwi Purnamasari
3. 13521027 / Agsha Athalla Nurkareem
4. 13521029 / M Malik I Baharsyah
5. 13521030 / Jauza Lathifah Annassalafi
Asisten Pembimbing : Aditya Bimawan

DAFTAR ISI

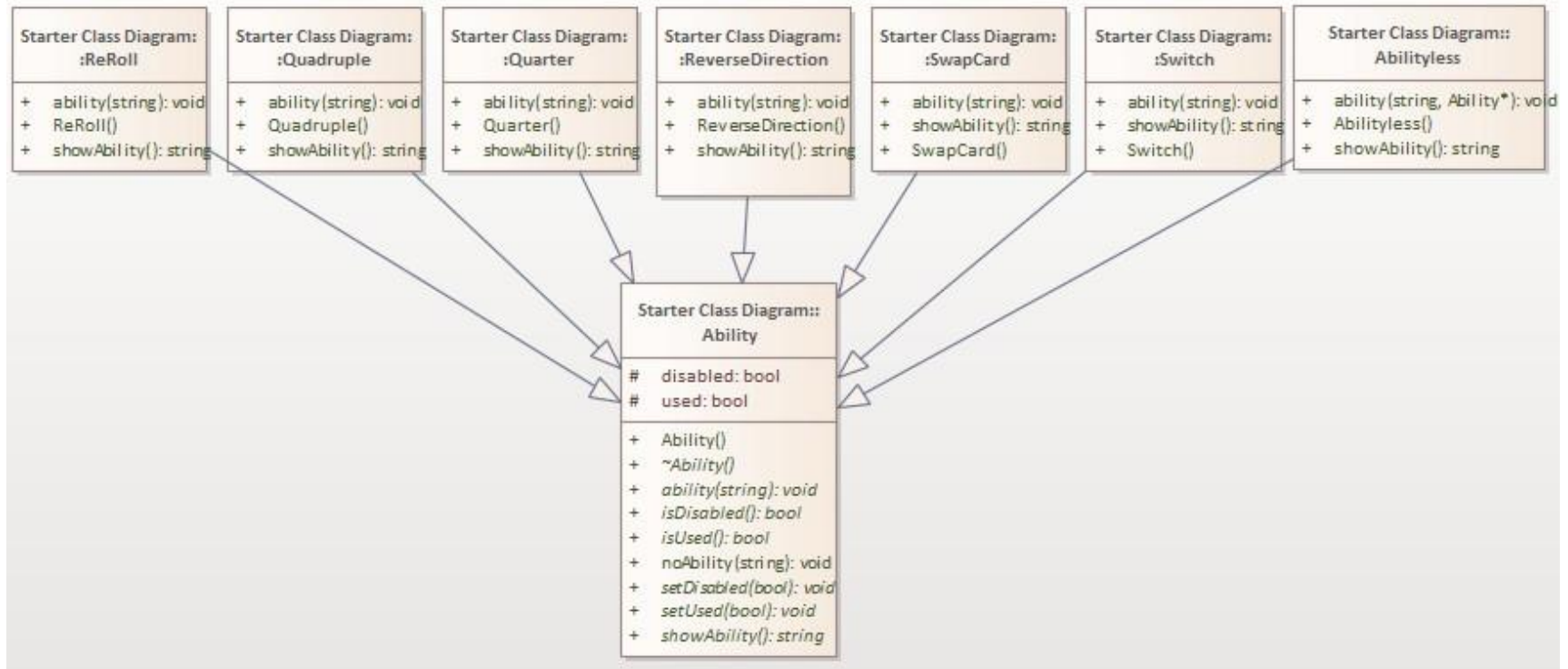
1. Diagram Kelas	2
1.1. Hierarki Class	2
1.2. Class Ability	3
1.3. Class InventoryHolder	4
1.4. Class Value	5
1.5. Class Valueable	6
1.6. Class Exception	7
2. Penerapan Konsep OOP	8
2.1. Inheritance & Polymorphism	8
2.2. Method/Operator Overloading	10
2.3. Template & Generic Classes	15
2.4. Exception	16
2.5. C++ Standard Template Library	19
2.6. Abstract Base Class (Konsep OOP lain)	21
3. Pembagian Tugas	22
Lampiran	25

1. Diagram Kelas

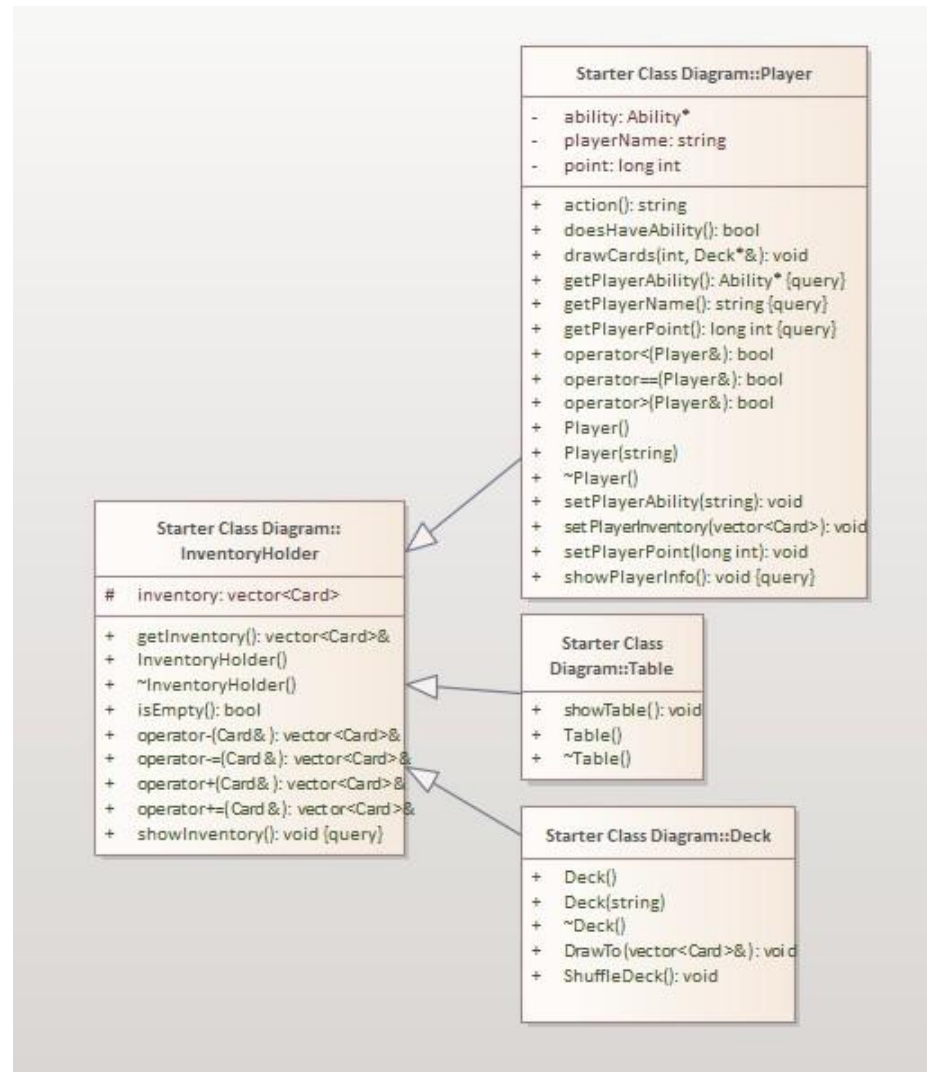
1.1. Hierarki Class



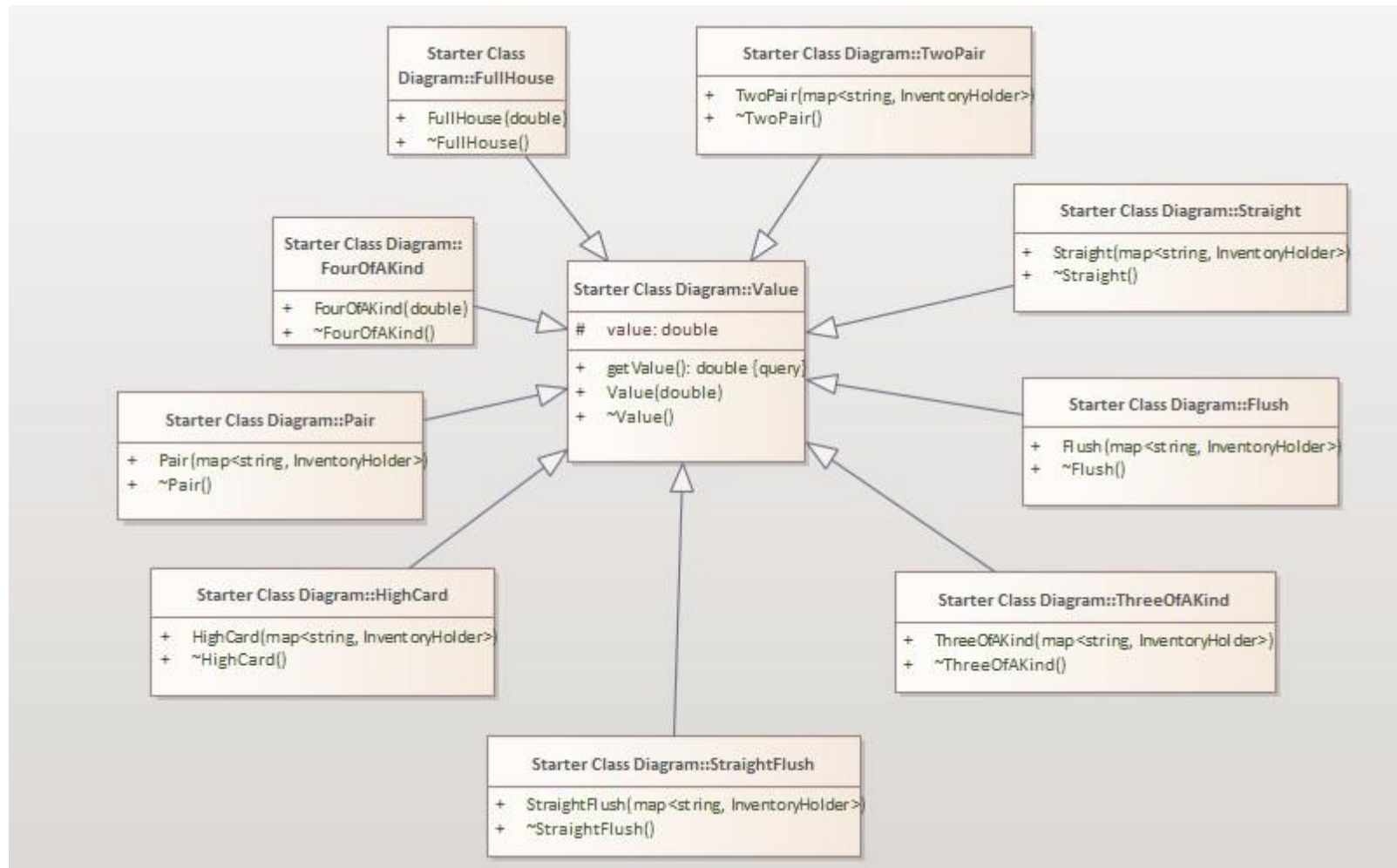
1.2. Class Ability



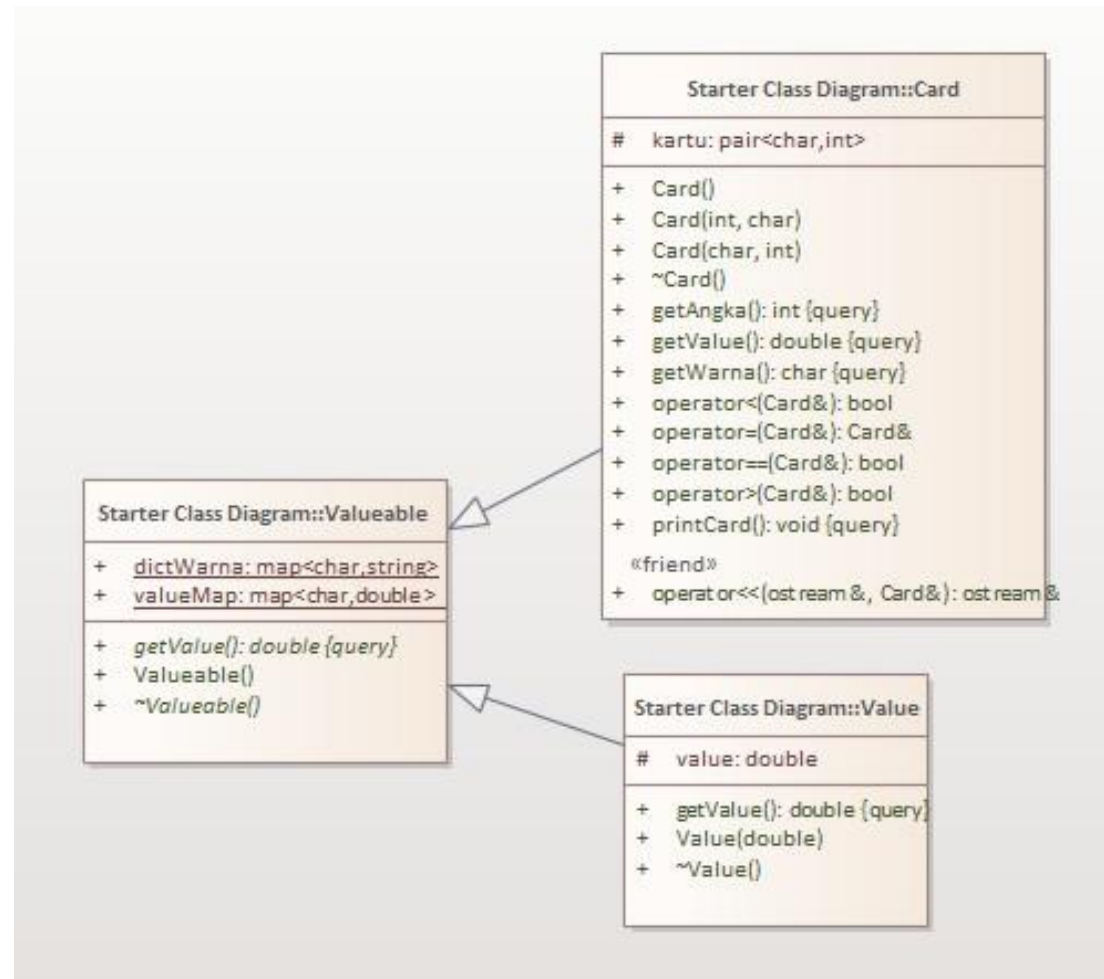
1.3. Class InventoryHolder



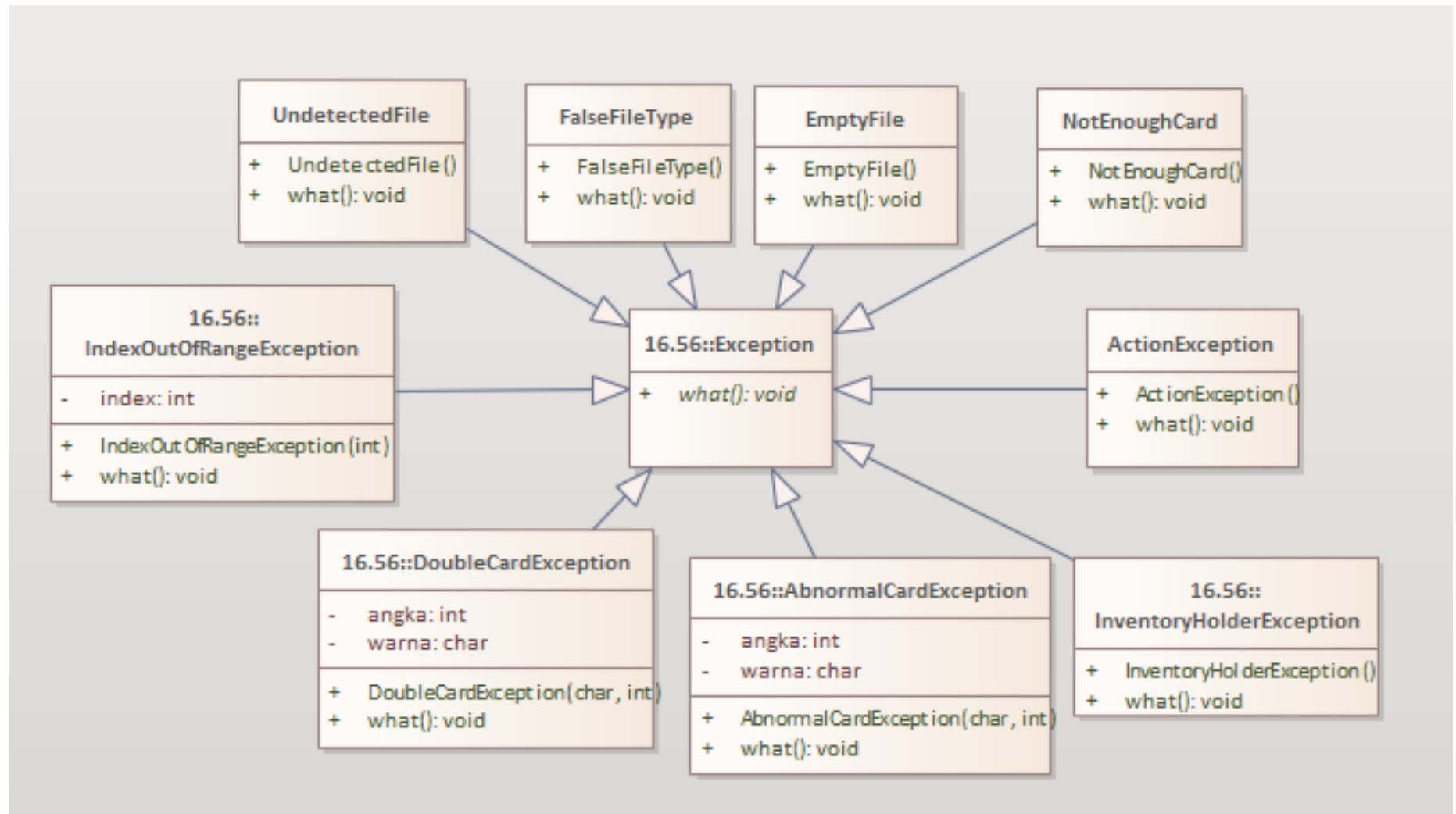
1.4. Class Value



1.5. Class Valueable



1.6. Class Exception



1.7. Penjelasan Desain Class

Dalam pengerjaan tugas besar ini, kami mengimplementasikan beberapa konsep inheritance pada kelas-kelas seperti InventoryHolder, Ability, Exception, dan Valuable agar dapat memanfaatkan konsep polymorphism. Class Exception akan membuat satu class utama dan nantinya akan membuat subclass lainnya yang akan dipanggil dalam fungsi try, catch, throw sehingga meminimalisir penggunaan handler pada setiap tipe exception yang berakibat banyaknya baris kode. Kekurangan dari pendekatan ini pada saat pengimplementasian class adalah pada class ability tidak dapat langsung terhubung dengan class GameState sehingga permainan berjalan dengan kurang modular.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep inheritance dan polymorphism diterapkan di beberapa kelas pada program, yaitu pada kelas Card dan Value yang meng-inherit kelas Valuable, kelas Deck, Player, dan Table meng-inherit kelas InventoryHolder, kelas Flush, FourOfAKind, FullHouse, HighCard, Pair, Straight, StraightFlush, ThreeOfAKind, dan TwoPair yang meng-inherit kelas Value, dan Player, ReRoll, Quadruple, Quarter, ReverseDirection, SwapCard, Switch, dan Abilityless yang meng-inherit kelas Ability. Konsep inheritance diterapkan untuk mengabstraksikan kelas yang terbentuk dan juga memungkinkan untuk penerapan polymorphism dalam pemrograman berorientasi objek. Polymorphism diterapkan untuk memudahkan dalam pengolahan suatu objek dengan hanya cukup menggunakan tipe kelas dasarnya saja. Implementasi inheritance pada kelas card dan value memiliki tujuan valuable card maupun value dapat diletakkan dalam slot dan diperlakukan sebagai agai kelas yang sama (Polymorphism) yaitu kelas Valuable. Kelas InventoryHolder dan Valueable memiliki tipe kelas anak yang beragam, dimana polymorphism dibutuhkan untuk mengabstraksi tipe deklarasi objek-objek menjadi tipe kelas dasarnya tersebut ketika digunakan pada suatu tipe data collection (seperti array, list, dan vector).

Kelas Card dan Value yang meng-inherit kelas Valueable

```

1  class Valueable{
2  public:
3      Valueable(){};
4      virtual ~Valueable(){}; //?? this stucked the prog
5      static map<char,double> valueMap;
6      static map<char,string> dictWarna;
7      virtual double getValue() const = 0;
8  };

```

```

1  class Card : public Valueable {
2  protected:
3      pair<char,int> kartu; //pair warna dan angka
4  public:
5      //ctor, dtor
6      Card();
7      Card(int,char);
8      Card(char,int);
9      ~Card();
10     char getWarna() const;
11     int getAngka() const;
12     double getValue() const;
13     void printCard() const;
14     bool operator<(const Card&);
15     bool operator>(const Card&);
16     bool operator==(const Card&);
17     friend ostream& operator<< (ostream& os, const Card& card);
18 };

```



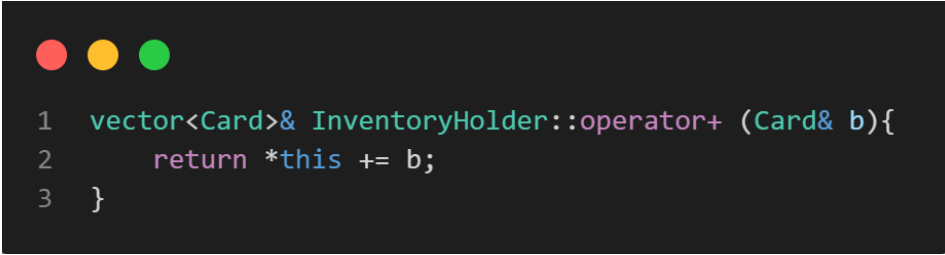
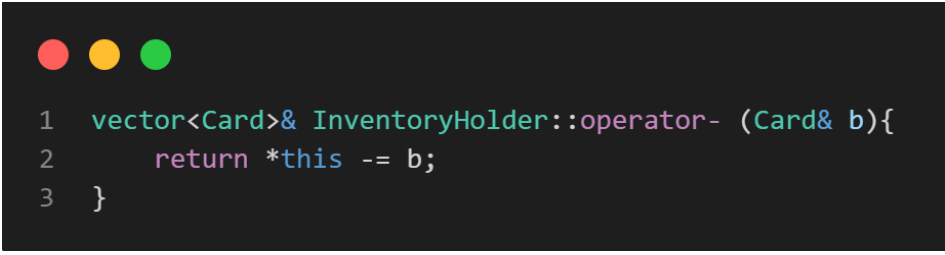
```
1  class Value : public Valueable {  
2  protected:  
3      double value;  
4  public:  
5      // Value(Combination* co);  
6      Value(double);  
7      ~Value(){};  
8      double getValue() const;  
9  };
```

2.2. Method/Operator Overloading

Konsep *Method/Operator Overloading* diimplementasikan pada beberapa aspek program. *Function overloading* sendiri adalah fasilitas yang memungkinkan nama fungsi yang sama dapat dipanggil dengan jenis dan jumlah parameter (*function signature*) yang berbeda-beda. Fungsi operator dapat digunakan dalam operasi aritmatika terhadap objek-objek matematika agar lebih terlihat alami dan mudah dipahami oleh pembaca program. Selain itu, fungsi operator dapat menciptakan operasi input/output yang seragam dengan memanfaatkan stream I/O dari C++. Pada program ini *operator overloading* digunakan dalam kelas *Card*, *InventoryHolder*, dan *Player*. Pada kelas *InventoryHolder*, operator yang digunakan ialah "+=", "-=", "+", dan "-".

Operator overloading pada InventoryHolder.cpp

<p>operator+=(Card&)</p> <p>Untuk menambahkan suatu Card ke dalam inventory</p>	 <pre> 1 vector<Card>& InventoryHolder::operator+=(Card& input){ 2 this->inventory.push_back(input); 3 return this->inventory; 4 }</pre>
<p>operator-=(Card&)</p> <p>Untuk menghapus suatu Card di dalam inventory</p>	 <pre> 1 vector<Card>& InventoryHolder::operator-=(Card& input){ 2 for (int i = 0; i < this->inventory.size(); i++) 3 { 4 if (this->inventory[i] == input){ 5 this->inventory.erase(this->inventory.begin() + i); 6 break; 7 } else if (i == this->inventory.size()-1){ 8 //throw notFoundException; 9 cout << "T not found" << endl; 10 } 11 } 12 return this->inventory; 13 }</pre>

<p>operator+(Card&)</p> <p>Untuk menambahkan suatu Card ke dalam inventory</p>	 <pre> 1 vector<Card>& InventoryHolder::operator+ (Card& b){ 2 return *this += b; 3 } </pre>
<p>operator-(Card&)</p> <p>Untuk menghapus suatu Card di dalam inventory</p>	 <pre> 1 vector<Card>& InventoryHolder::operator- (Card& b){ 2 return *this -= b; 3 } </pre>

Method overloading digunakan karena terdapat satu fungsi yang memiliki *behaviour* yang berbeda pada suatu fungsi yang sama. Pengimplementasian konsep ini terdapat pada kelas *ability* dan kelas *inherit*-nya. Pada kelas *ability* terdapat prosedur *ability* dan fungsi *showAbility* yang bersifat *virtual* dan diturunkan kepada *inherit*-nya. Pada tiap kelas *inherit*-nya, fungsi dan prosedur ini memiliki *behaviour* dan pengimplementasian yang berbeda.

Method overloading pada *Ability.cpp*

class ReRoll	<pre>1 void ReRoll::ability(string input){ 2 string valid = "REROLLRE-ROLL"; 3 if (valid.find(input) != -1){ 4 cout << "RE-ROLL ability activated" << endl; 5 } else { 6 noAbility(input); 7 } 8 }</pre>
class Quadruple	<pre>1 void Quadruple::ability(string input){ 2 string valid = "QUADRUPLE"; 3 if (valid.find(input) != -1){ 4 cout << "QUADRUPLE ability activated" << endl; 5 } else { 6 noAbility(input); 7 } 8 }</pre>
class Quarter	<pre>1 void Quarter::ability(string input){ 2 string valid = "QUARTER"; 3 if (valid.find(input) != -1){ 4 cout << "QUARTER ability activated" << endl; 5 } else { 6 noAbility(input); 7 } 8 }</pre>

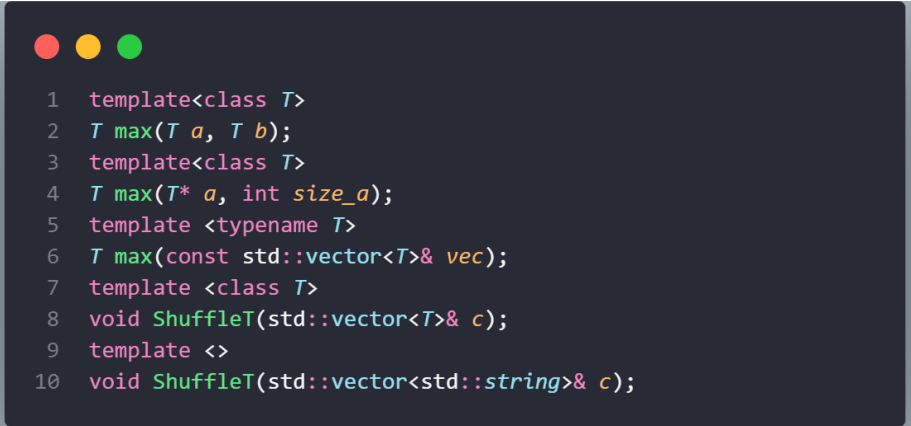
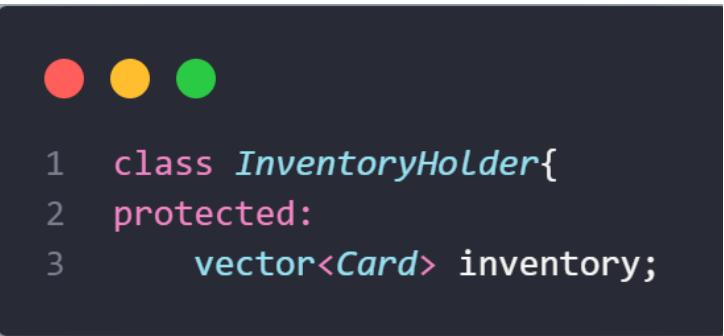
class ReverseDirection	<pre>1 void ReverseDirection::ability(string input){ 2 string valid = "REVERSEDIRECTION"; 3 if (valid.find(input) != -1){ 4 cout << "REVERSE ability activated" << endl; 5 } else { 6 noAbility(input); 7 } 8 }</pre>
class SwapCard	<pre>1 void SwapCard::ability(string input){ 2 string valid = "SWAPCARD"; 3 if (valid.find(input) != -1){ 4 cout << "SWAP ability activated" << endl; 5 } else { 6 noAbility(input); 7 } 8 }</pre>
class Switch	<pre>1 void Switch::ability(string input){ 2 string valid = "SWITCH"; 3 if (valid.find(input) != -1){ 4 cout << "SWITCH ability activated" << endl; 5 } else { 6 noAbility(input); 7 } 8 }</pre>

class Abilityless

```
1 void Abilityless::ability(string input, Ability* targetAb){
2     string valid = "ABILITYLESS";
3     if (valid.find(input) != -1){
4         cout << "ABILITYLESS ability activated" << endl;
5         delete targetAb;
6         targetAb = new Ability;
7     } else if (valid.find(input) == -1){
8         noAbility(input);
9     } else if (targetAb == nullptr || targetAb->showAbility() == "NONE"){
10         cout << "Target tidak memiliki ability" << endl;
11     }
12 }
```

2.3. Template & Generic Classes

Template adalah mekanisme dalam C++ yang memungkinkan untuk menulis kode yang dapat diterapkan pada tipe data yang berbeda. Pada Tugas besar ini, Konsep Template diimplementasikan pada beberapa aspek program, seperti pada generics dan inventory holder. Generic Class digunakan untuk mendefinisikan kelas dasar dengan tipe kelas agregat objek yang memiliki tipe yang sama. Dengan Template dan Generic Class, Ketika akan membuat suatu class ataupun method, kita tidak perlu melakukan method overloading berulang kali, sehingga dapat mengurangi redudansi dalam pembuatan suatu method yang berulang. Hal ini dapat mempercepat proses pengembangan dan mengurangi jumlah kesalahan yang mungkin terjadi karena tidak perlu menulis ulang kode yang sama. Penggunaan generic class dan template pada pengerjaan tugas ini diantaranya sebagai berikut.

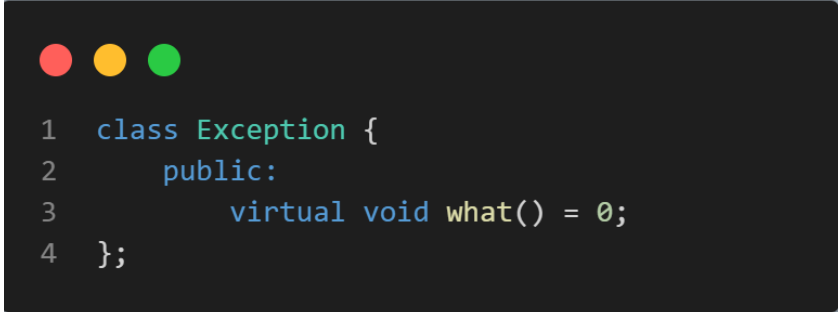
T sebagai tipe bentukan pada fungsi max dan shuffleT	 <pre> 1 template<class T> 2 T max(T a, T b); 3 template<class T> 4 T max(T* a, int size_a); 5 template <typename T> 6 T max(const std::vector<T>& vec); 7 template <class T> 8 void ShuffleT(std::vector<T>& c); 9 template <> 10 void ShuffleT(std::vector<std::string>& c); </pre>
Card pada pembentukan inventory	 <pre> 1 class InventoryHolder{ 2 protected: 3 vector<Card> inventory; </pre>

2.4. Exception

Konsep Exception merupakan suatu mekanisme yang digunakan untuk menangani situasi ketika program mengalami kesalahan atau kondisi tidak terduga. Konsep ini hampir digunakan di banyak kelas karena banyak kasus yang perlu ditangani sehingga membutuhkan exception. Dengan menggunakan fitur exception bersama dengan sintaks try, catch, dan throw, kita dapat mengurangi kesalahan akibat ketidaktepatan pada implementasi. Ketika sebuah kesalahan terjadi, exception akan menghentikan eksekusi program pada titik tersebut dan mencari blok

code yang sesuai dengan tipe kesalahan yang terjadi. Jika blok code tersebut ditemukan, maka program akan dijalankan sesuai dengan instruksi yang ada di dalam blok code tersebut. Penggunaan Exception dalam pengerjaan tugas ini diantaranya sebagai berikut.

Abstract Base Class
"Exception"



```
1 class Exception {  
2     public:  
3         virtual void what() = 0;  
4 };
```



Beberapa bentuk Exception

```
1 class IndexOutOfRangeException : public Exception {
2     private:
3         int index;
4     public:
5         IndexOutOfRangeException(int index){
6             this->index = index;
7         }
8         void what(){
9             cout << this->index << " berada di luar jangkauan" << endl;
10        }
11    };
12
13 class DoubleCardException : public Exception {
14     private:
15         char warna;
16         int angka;
17     public:
18         DoubleCardException(char warna, int angka){
19             this->warna = warna;
20             this->angka = angka;
21         }
22         void what(){
23             cout << "Gagal karena terdapat 2 kartu dengan angka " << this->angka << " dan warna " << this->warna <<
24             " yang sama";
25        }
26    };
```


2.5. C++ Standard Template Library

Konsep C++ Standard Template Library memanfaatkan STL dari library C++ pada beberapa kelas dalam program untuk menyimpan data pada struktur data yang sudah disediakan sehingga meningkatkan keefisienan saat digunakan, sedangkan penggunaan STL bertujuan untuk menggunakan fungsi-fungsi yang tersedia dalam library STL sehingga tidak perlu diimplementasikan sendiri. Pada kelas *Card*, digunakan `pair<char, int>` untuk menyimpan kartu dengan warna dan angka tertentu. Pada beberapa kelas juga digunakan STL map, salah satunya pada kelas *Combination* digunakan `map<string, InventoryHolder>` untuk menyimpan *inventory* kartu dengan parameter string tertentu. Selain itu, STL yang digunakan pada program ini antara lain vector dan deque.

Beberapa cuplikan penggunaan STL yang terdapat dalam program.


Penggunaan STL pair pada kelas <i>Card</i>	 <pre> 1 class Card : public Valueable { 2 protected: 3 pair<char,int> kartu; //pair warna dan angka </pre>
Penggunaan STL map pada kelas <i>Combination</i>	 <pre> 1 class Combination { 2 private: 3 InventoryHolder hold; // kombinasi kartu 4 map<string, InventoryHolder> mapcard; 5 Value *value; // pointer value untuk menghitung nilai kombinasi </pre>

Penggunaan STL deque dan vector pada kelas GameState



```
1 class GameState{
2 private:
3     static deque<int> turn;
4     static int turnStartFrom;
5     static bool reverseTurn;
6     int round,playerCount;
7     vector<string> abilities;
8     long long int prize;
9     Deck* cardDeck;
10    Table* table;
11    Player* players;
```

Penggunaan STL vector pada kelas InventoryHolder



```
1 class InventoryHolder{
2 protected:
3     vector<Card> inventory;
```

2.6. Abstract Base Class (Konsep OOP lain)

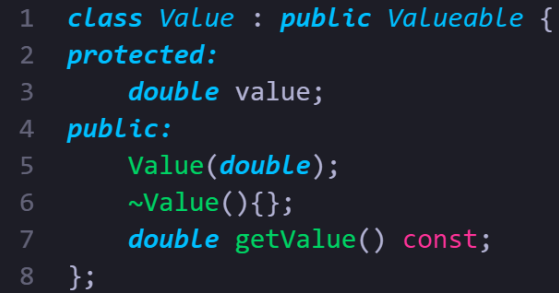
Dalam pengerjaan tugas besar, kami menerapkan konsep Abstract Base Class ke dalam beberapa class, seperti Valuable Class. Abstract Base Class merupakan suatu cara untuk memetakan suatu class dengan membuat sebuah kelas abstrak sebagai basis dalam membentuk kelas-kelas turunannya. Dengan menggunakan konsep abstract base class, kita dapat membuat sebuah kelas untuk mengimplementasikan method-method tertentu, sehingga memastikan kelas tersebut sesuai dengan kebutuhan aplikasi. Dalam kasus ini, valueable dibuat sebagai abstract base class untuk kemudian dibuat inheritance value yang akan dipakai dalam class-class combination.

Berikut ini merupakan beberapa contoh penggunaan Abstract Base Class dalam program.

Abstract Base Class Valuable

```
1  class Valuable{
2  public:
3      Valuable(){};
4      virtual ~Valuable(){};
5      static map<char,double> valueMap;
6      static map<char,string> dictWarna;
7      virtual double getValue() const = 0;
8  };
```

Class value yang mengikuti
Abstract Base Class Valuable



```

1  class Value : public Valueable {
2  protected:
3      double value;
4  public:
5      Value(double);
6      ~Value(){};
7      double getValue() const;
8  };

```

3. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Deck	13521009	13521009, 13521018, 13521027, 13521029, 13521030
Player Card & Table Card	13521009, 13521018, 13521027, 13521029, 13521030	13521009, 13521018, 13521027, 13521029, 13521030
Putaran Permainan	13521009, 13521029	13521009, 13521018, 13521027, 13521029, 13521030
Poin Hadiah	13521009, 13521029	13521009, 13521018, 13521027, 13521029, 13521030

Perintah Double and Half	13521009, 13521029	13521009, 13521018, 13521027, 13521029, 13521030
Perintah Next	13521009	13521009, 13521018, 13521027, 13521029, 13521030
Ability	13521009, 13521018, 13521027, 13521029, 13521030	13521009, 13521018, 13521027, 13521029, 13521030
Combination	13521018, 13521027, 13521029, 13521030	13521009, 13521018, 13521027, 13521029, 13521030

Foto Kelompok:



Lampiran Link github:

https://github.com/christodharma/IF2210_TB1_911.git

Lampiran Asistensi :

Kode Kelompok : 911

Nama Kelompok : ninuninu

1. 13521009 / Christophorus Dharma Winata
2. 13521018 / Syarifa Dwi Purnamasari
3. 13521027 / Agsha Athalla Nurkareem
4. 13521029 / M Malik I Baharsyah
5. 13521030 / Jauza Lathifah Annassalafi

Asisten Pembimbing : Aditya Bimawan

1. Konten Diskusi

Pada awal asistensi pertama, setelah membaca spesifikasi tugas besar kami belum mengerti mengenai bagaimana cara kerja permainan kompetisi kartu ini, sehingga kemudian asisten memberikan penjelasan mengenai garis besar permainan tersebut. Kemudian kelompok kami memberikan class diagram sementara untuk dimintai feedback hal apa saja yang sekiranya perlu diubah dalam class diagram tersebut.

2. Tindak Lanjut

- Mengikuti saran yang diberikan oleh kakak asisten
- Membuat program sesuai dengan yang sudah dijelaskan oleh kakak asisten